An LTL Proof System for Runtime Verification

Clare Cini[1] and Adrian Francalanza[1]

Computer Science, ICT, University of Malta
{clare.cini.08,adrian.francalanza}@um.edu.mt

Abstract. We propose a local proof system for LTL formalising deductions within the constraints of Runtime Verification (RV), and show how such a system can be used as a basis for the construction of online runtime monitors. Novel soundness and completeness results are proven for this system. We also prove decidability and incrementality properties for a monitoring algorithm constructed from it. Finally, we relate its expressivity to existing symbolic analysis techniques used in RV.

1 Introduction

Runtime verification (RV) is a lightweight verification technique that checks whether the current execution of a system under scrutiny satisfies or violates a given correctness property. It has its origins in model checking, as a more scalable (yet still formal) approach to program verification where state explosion problems (which are part and parcel of model checking) are mitigated [LS09]. Linear Temporal Logic, (LTL) [Pnu77] is prevalently used for formal expositions of RV [Gei01,SRA04,BLS07,BLS10,BLS11,BF12], because it has a pleasingly straightforward definition over strings, denoting execution traces.

Proof systems [Bus98,TS00] embody mechanical syntactic deductions similar to those made by monitors in RV. We propose a proof system for LTL attuned to the constraints of an RV setting, and show how it can be used as a basis for monitor construction. Although deductive systems for LTL exist, *e.g.*, [MP91,KI11,BL08] they are geared towards reasoning about the full statespace of a system. By contrast, our proof system is *local* [SW91] focussing on checking whether a specific point lies within a property set, instead of interpreting a formula *wrt.* a set of points; this mirrors closely the runtime analysis in RV.

RV settings pose further constraints on our symbolic analysis. In online settings, deductions are often performed on the *partial* traces generated thus far, while the program is still executing. This has two important consequences: (*a*) *conclusive* deductions must be *consistent with any extension* leading to a complete trace (*b*) in order to keep RV overheads low, *inconclusive* deductions must reusable, and contribute to deductions of subsequent extensions *i.e.,* the analysis must be *incremental*. In addition, monitors for partial traces typically reason about trace *satisfactions*, but also trace *violations* [BLS11,BLS10] so as to determine good/bad prefixes [KYV01]. Accordingly, proof system deductions should reason directly about both satisfactions and violations. Moreover, timely detections often require *synchronous* monitor instrumentation where monitored

system execute in lock-step with the respective monitor, producing a trace event and waiting for the monitor to terminate its (incremental) analysis before executing further. Thus, in order for such an instrumentation to be safe, it is important to ensure that incremental deductions are *decidable*.

We formally compare our system with other LTL symbolic techniques used in RV. In particular we consider Geilen's work [Gei01], based on informative prefixes [KYV01], as well as Sen *et al.*'s work [SRA04] which is based on derivatives [HR01]. Apart from enabling a better understanding of each approach, facilitating comparisons between seemingly different formalisations, this study enables cross fertilisation of techniques from one formalisation to the other.

The paper is structured as follows. After introducing the logic, §2, we present our proof system in §3. §4 presents the associated monitor algorithm. §5 details formal comparisons with other symbolic analyses and §6 concludes.

2 The Logic: An LTL Primer

Syntax. Fig. 1 defines the *core* syntax of LTL as used in [BLS11,EFH$^+$03], parameterised by a set of predicates $p \in \textsc{Pred}$. It consists of two base cases, *i.e.*, the true formula, tt, and a predicate formula, p, standard negation and conjunction constructors, $\neg\psi$ and $\psi_1 \wedge \psi_2$, and the characteristic *next* and *until* formulas, $\mathsf{X}\psi$ and $\psi_1 \ \mathsf{U} \ \psi_2$ *resp.* Other studies of LTL (*e.g.*, [Gei01,BL08]) prefer to work with formulas in *negation normal form* (nnf), where negations are pushed to the leaves of a formula. To accommodate this, we also consider an extended LTL syntax in Fig. 1, that also includes base formulas for falsity, ff, and constructors such as disjunctions, $\varphi_1 \vee \varphi_2$, and release formulas, $\varphi_1 \ \mathsf{R} \ \varphi_2$. Our extended syntax also employs an extended predicate notation that includes *co-predicates*, *i.e.*, for any predicate[1] $p = S \subseteq \Sigma$, its co-predicate, denoted as $\overline{p}$, represents its *dual* and is defined as $\Sigma \setminus S$. This allows us to eliminate negations from normalised formulas; because of this we sometimes refer to an nnf formula as *negation-free*. Fig. 1 also defines a translation function, $\langle - \rangle :: \text{LTL} \to \textsc{eLtl}$ from formulas of the core LTL to a negation-free formula in the extended syntax.

Model. The logic semantics is also given in Fig. 1. It assumes an alphabet, Σ (with element variables σ), over which predicates are defined, $p :: \Sigma \to \textsc{Bool}$. As in other RV studies [Gei01,SRA04,BLS11], the logic is defined over *infinite* strings, $s \in \Sigma^\omega$; *finite* strings over the same alphabet are denoted by the variable $t \in \Sigma^*$. A string with element σ at its head is denoted as σs (*resp.* σt). For indexes $i, j \in \textsc{Nat}$, s_i denotes the i^{th} *element* in the string (starting from index 0) and $[s]^i$ denotes the *suffix* of s starting at index i; note that for any s, $[s]^0 = s$. Infinite strings with a regular (finite) pattern t are sometimes denoted as t^*, whereas the shorthand $t\ldots$ represents infinite strings with a (finite) prefix t.

Semantics. The denotational semantic function $[\![-]\!] :: \textsc{eLtl} \to \mathcal{P}(\Sigma^\omega)$ is defined by induction over the structure of LTL formulas; in Fig. 1 we define the

[1] Predicates are sometimes denoted as sets over Σ.

Core LTL Syntax

$$\psi \in \text{LTL} \ ::= \ \mathsf{tt} \mid p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \mathsf{X}\psi \mid \psi_1 \ \mathsf{U} \ \psi_2$$

Extended LTL Syntax

$$\begin{aligned} \varphi \in \text{eLTL} \ ::= \ & \mathsf{tt} \mid p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \ \mathsf{U} \ \varphi_2 \mid \mathsf{X}\varphi \mid \neg\varphi \\ \mid \ & \mathsf{ff} \mid \overline{p} \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \ \mathsf{R} \ \varphi_2 \end{aligned}$$

Formula Translation (Normalisation)

$$\langle \mathsf{tt} \rangle \stackrel{\text{def}}{=} \mathsf{tt} \qquad \langle \neg\mathsf{tt} \rangle \stackrel{\text{def}}{=} \mathsf{ff} \qquad \langle p \rangle \stackrel{\text{def}}{=} p \qquad \langle \neg p \rangle \stackrel{\text{def}}{=} \overline{p}$$

$$\langle \mathsf{X}\psi \rangle \stackrel{\text{def}}{=} \mathsf{X}\langle \psi \rangle \qquad \langle \neg\mathsf{X}\psi \rangle \stackrel{\text{def}}{=} \mathsf{X}\langle \neg\psi \rangle \qquad \langle \neg\neg\psi \rangle \stackrel{\text{def}}{=} \langle \psi \rangle$$

$$\langle \psi_1 \wedge \psi_2 \rangle \stackrel{\text{def}}{=} \langle \psi_1 \rangle \wedge \langle \psi_2 \rangle \qquad \langle \neg\big(\psi_1 \wedge \psi_2\big) \rangle \stackrel{\text{def}}{=} \langle \neg\psi_1 \rangle \vee \langle \neg\psi_2 \rangle$$

$$\langle \psi_1 \ \mathsf{U} \ \psi_2 \rangle \stackrel{\text{def}}{=} \langle \psi_1 \rangle \ \mathsf{U} \ \langle \psi_2 \rangle \qquad \langle \neg\big(\psi_1 \ \mathsf{U} \ \psi_2\big) \rangle \stackrel{\text{def}}{=} \langle \neg\psi_1 \rangle \ \mathsf{R} \ \langle \neg\psi_2 \rangle$$

Semantics

$$\begin{aligned} [\![\mathsf{tt}]\!] &\stackrel{\text{def}}{=} \Sigma^\omega & [\![\mathsf{ff}]\!] &\stackrel{\text{def}}{=} \emptyset \\ [\![p]\!] &\stackrel{\text{def}}{=} \{s \mid p(s_0)\} & [\![\overline{p}]\!] &\stackrel{\text{def}}{=} \{s \mid \text{not } p(s_0)\} \\ [\![\varphi_1 \wedge \varphi_2]\!] &\stackrel{\text{def}}{=} [\![\varphi_1]\!] \cap [\![\varphi_1]\!] & [\![\varphi_1 \vee \varphi_2]\!] &\stackrel{\text{def}}{=} [\![\varphi_1]\!] \cup [\![\varphi_1]\!] \\ [\![\mathsf{X}\psi]\!] &\stackrel{\text{def}}{=} \big\{s \mid [s]^1 \in [\![\psi]\!]\big\} & [\![\neg\varphi]\!] &\stackrel{\text{def}}{=} \big(\Sigma^\omega\big) \setminus [\![\varphi]\!] \\ [\![\varphi_1 \ \mathsf{U} \ \varphi_2]\!] &\stackrel{\text{def}}{=} \big\{s \mid \exists j \text{ such that } [s]^j \in [\![\varphi_2]\!] \text{ and } (i < j \text{ implies } [s]^i \in [\![\varphi_1]\!])\big\} \\ [\![\varphi_1 \ \mathsf{R} \ \varphi_2]\!] &\stackrel{\text{def}}{=} \big\{s \mid \forall j \text{ we have } \big([s]^j \in [\![\varphi_2]\!] \text{ or } (\exists i < j \text{ such that } [s]^i \in [\![\varphi_1]\!])\big)\big\} \end{aligned}$$

Fig. 1. Linear Temporal Logic Syntax and Semantics

semantics for the extended LTL syntax (of which the core syntax is a subset). Most cases are standard. For instance, $[\![\mathsf{tt}]\!]$ (*resp.* $[\![\mathsf{ff}]\!]$) returns the universal (*resp.* empty) set of strings, $[\![\neg\varphi]\!]$ returns the dual of $[\![\varphi]\!]$, whereas $[\![\varphi_1 \wedge \varphi_2]\!]$ (*resp.* $[\![\varphi_1 \vee \varphi_2]\!]$) denotes the intersection (*resp.* union) of the meaning of its subformulas, $[\![\varphi_1]\!]$ and $[\![\varphi_2]\!]$. The meaning of $[\![p]\!]$ (*resp.* $[\![\overline{p}]\!]$) contains all strings whose *first element* satisfies the predicate p (*resp.* $\overline{p}$). The temporal formulas are more involving. The denotation of $[\![\mathsf{X}\varphi]\!]$ contains all strings whose immediate suffix (*i.e.,* at index 1) is included in $[\![\psi]\!]$. Until formulas $[\![\varphi_1 \ \mathsf{U} \ \varphi_2]\!]$ contain all strings that contain a suffix (at *some* index j) satisfying $[\![\psi_2]\!]$, and *all* the suffixes preceding j satisfy $[\![\psi_1]\!]$. Finally, release formulas, $\varphi_1 \ \mathsf{R} \ \varphi_2$ contain strings whose suffixes *always* satisfy φ_2, as well as strings that contain a suffix satisfying both φ_1 and φ_2 and *all* the preceding suffixes satisfying φ_2.

The denotational semantics allows us to observe the duality between the formulas tt, $\varphi_1 \wedge \varphi_2$ and $\psi_1 \ \mathsf{U} \ \psi_2$, and their counterparts ff, $\varphi_1 \vee \varphi_2$ and $\psi_1 \ \mathsf{R} \ \psi_2$. It also helps us understand the mechanics of the translation function, pushing negation to the leaves of a formula using negation propagation identities (*e.g.,* DeMorgan's law), converting constructors to their dual constructor; at the leaves the function then performs direct translations from tt and p to ff and $\overline{p}$

resp. The semantics also allows us to prove Prop. 1, justifying the use of a corresponding negation-free formula instead of a core LTL formula, in order to reason exclusively in terms of positive interpretations. It states that (*i*) the translation function is *total* (otherwise the equality cannot be determined) but also that (*ii*) the translated formula *preserves the semantic meaning* of the original formula.

Proposition 1. *For any* $\psi \in \text{LTL}$, $[\![\psi]\!] = [\![\langle\psi\rangle]\!]$

Example 1. The behaviour of a traffic-light system may be described by observing its states consisting of green, g, orange, o, and red, r. Complete executions may thus be represented as traces (strings) over the alphabet $\Sigma = \{g, o, r\}$. Predicate definitions may be defined as sets over this alphabet Σ *e.g.,* $\mathsf{st} = \{o, r\}$ is true for *stopping* actions o and r; singleton-set predicates are denoted by single-letter names *e.g.,* $\mathsf{g} = \{g\}$. We can specify the following properties:

- $(\neg\mathsf{r}) \wedge \mathsf{X}\mathsf{r}$ describes a trace where the system is not in a red state initially, but turns red at the next instant. *e.g., gr* ... or *or* ...;
- $\mathsf{g}\ \mathsf{U}\ \mathsf{o}$ describes traces that eventually switch to the orange state from a green state *e.g., go* ... or *gggo* ...;
- $\mathsf{G}\,\mathsf{st}$, *i.e.,* always st, which is shorthand for $\neg(\mathsf{tt}\ \mathsf{U}\ \neg\mathsf{st})$, describes traces that contain only stopping states, *e.g.,* strings of the form $(or)^*$ or r^*.

To determine whether $r^* \in [\![\mathsf{G}\,\mathsf{st}]\!]$, we can use $[\![\neg(\mathsf{tt}\ \mathsf{U}\ \neg\mathsf{st})]\!]$ for which we would need to calculate $[\![\mathsf{tt}\ \mathsf{U}\ \neg\mathsf{st}]\!]$ and then take its dual. Alternatively, we can calculate the denotation of $\langle\neg(\mathsf{tt}\ \mathsf{U}\ \neg\mathsf{st})\rangle$, which translates to $\mathsf{ff}\ \mathsf{R}\ \mathsf{st}$, and check inclusion *wrt.* the translated negation-free formula, safe in the knowledge that $[\![\neg(\mathsf{tt}\ \mathsf{U}\ \neg\mathsf{st})]\!] = [\![\mathsf{ff}\ \mathsf{R}\ \mathsf{st}]\!]$. Using similar reasoning, to determine whether $r \ldots \notin [\![(\neg\mathsf{r}) \wedge \mathsf{X}\mathsf{r}]\!]$, we can check whether $r \ldots \in [\![\mathsf{r} \vee \mathsf{X}\bar{\mathsf{r}}]\!]$ holds. ■

3 An Online Monitoring Proof System

Online[2] runtime verification of LTL properties consist in determining whether the current execution satisfies (or violates) a property from *the trace generated thus far*. We present a local proof system [SW91,BS92] that characterises such runtime analysis, and allows us to determine whether any (complete) trace ts with *finite* prefix t is included in (or excluded from) $[\![\varphi]\!]$. The proof system is defined as the least relation satisfying the rules in Fig. 2. These rules employ two, mutually dependent, judgements: the sequent $t \vdash^+ \varphi$ denotes a *satisfaction* judgement, whereas $t \vdash^- \psi$ denotes a *violation* judgement; note the polarity differentiating the two judgements, *i.e.,* $+$ and $-$.

Fig. 2 includes three satisfaction axioms (pTru, pPrd and pCoP) and three violation axioms (nFls, nPrd and nCoP); the system is parametric *wrt.* the pre-computation of predicates and co-predicates, p and $\overline{p}$. The conjunction and disjunction rules, pAnd, pOr1 and pOr2 (*resp.* nAnd1, nAnd2 and nOr) *decompose* the composite formula of the judgement for their premises. The negation

[2] By contrast, *offline* monitoring typically works on *complete* execution traces. [RH05]

Satisfaction Rules

$$\textsc{pTru}\frac{}{t \vdash^+ \mathsf{tt}} \qquad \textsc{pPrd}\frac{p(\sigma)}{\sigma t \vdash^+ p} \qquad \textsc{pCoP}\frac{\overline{p}(\sigma)}{\sigma t \vdash^+ \overline{p}} \qquad \textsc{pNeg}\frac{t \vdash^- \varphi}{t \vdash^+ \neg\varphi}$$

$$\textsc{pAnd}\frac{t \vdash^+ \varphi_1 \quad t \vdash^+ \varphi_2}{t \vdash^+ \varphi_1 \wedge \varphi_2} \qquad \textsc{pNxt}\frac{t \vdash^+ \varphi}{\sigma t \vdash^+ \mathsf{X}\varphi}$$

$$\textsc{pOr1}\frac{t \vdash^+ \varphi_1}{t \vdash^+ \varphi_1 \vee \varphi_2} \qquad \textsc{pOr2}\frac{t \vdash^+ \varphi_2}{t \vdash^+ \varphi_1 \vee \varphi_2}$$

$$\textsc{pUnt1}\frac{t \vdash^+ \varphi_2}{t \vdash^+ \varphi_1 \,\mathsf{U}\, \varphi_2} \qquad \textsc{pUnt2}\frac{\sigma t \vdash^+ \varphi_1 \quad t \vdash^+ \varphi_1 \,\mathsf{U}\, \varphi_2}{\sigma t \vdash^+ \varphi_1 \,\mathsf{U}\, \varphi_2}$$

$$\textsc{pRel1}\frac{t \vdash^+ \varphi_1 \quad t \vdash^+ \varphi_2}{t \vdash^+ \varphi_1 \,\mathsf{R}\, \varphi_2} \qquad \textsc{pRel2}\frac{\sigma t \vdash^+ \varphi_2 \quad t \vdash^+ \varphi_1 \,\mathsf{R}\, \varphi_2}{\sigma t \vdash^+ \varphi_1 \,\mathsf{R}\, \varphi_2}$$

Violation Rules

$$\textsc{nFls}\frac{}{t \vdash^- \mathsf{ff}} \qquad \textsc{nPrd}\frac{\overline{p}(\sigma)}{\sigma t \vdash^- p} \qquad \textsc{nCoP}\frac{p(\sigma)}{\sigma t \vdash^- \overline{p}} \qquad \textsc{nNeg}\frac{t \vdash^+ \varphi}{t \vdash^- \neg\varphi}$$

$$\textsc{nOr}\frac{t \vdash^- \varphi_1 \quad t \vdash^- \varphi_2}{t \vdash^- \varphi_1 \vee \varphi_2} \qquad \textsc{nNxt}\frac{t \vdash^- \psi}{\sigma t \vdash^- \mathsf{X}\psi}$$

$$\textsc{nAnd1}\frac{t \vdash^- \varphi_1}{t \vdash^- \varphi_1 \wedge \varphi_2} \qquad \textsc{nAnd2}\frac{t \vdash^- \varphi_2}{t \vdash^- \varphi_1 \wedge \varphi_2}$$

$$\textsc{nUnt1}\frac{t \vdash^- \varphi_1 \quad t \vdash^- \varphi_2}{t \vdash^- \varphi_1 \,\mathsf{U}\, \varphi_2} \qquad \textsc{nUnt2}\frac{\sigma t \vdash^- \varphi_2 \quad t \vdash^- \varphi_1 \,\mathsf{U}\, \varphi_2}{\sigma t \vdash^- \varphi_1 \,\mathsf{U}\, \varphi_2}$$

$$\textsc{nRel1}\frac{t \vdash^- \varphi_2}{t \vdash^- \varphi_1 \,\mathsf{R}\, \varphi_2} \qquad \textsc{nRel2}\frac{\sigma t \vdash^- \varphi_1 \quad t \vdash^- \varphi_1 \,\mathsf{R}\, \varphi_2}{\sigma t \vdash^- \varphi_1 \,\mathsf{R}\, \varphi_2}$$

Fig. 2. Satisfaction and Violation Proof Rules

rules pNeg and nNeg also decompose the formula, but *switch the modality* of the sequents for their premises, *transitioning* from one judgement form to the other. Specifically, in the case of pNeg, the satisfaction sequent $t \vdash^+ \neg\varphi$ is defined in terms of the *violation* sequent $t \vdash^- \varphi$ (and dually for nNeg).

The rules for the temporal formulas may decompose judgement formulas, *e.g.*, pUnt1, pRel1, nUnt1, nRel1, but may also analyse suffixes of the trace *in incremental fashion*. For instance, in order to prove $\sigma t \vdash^+ \mathsf{X}\varphi$, rule pNxt requires the satisfaction judgement to hold for the *immediate* suffix t and the subformula φ, *i.e.*, $t \vdash^+ \varphi$. Similarly, to prove the satisfaction sequent $\sigma t \vdash^+ \varphi_1 \,\mathsf{U}\, \varphi_2$, rule pUnt2 requires a satisfaction proof of the current trace σt and the subformula φ_1, as well as a satisfaction proof of the immediate suffix t *wrt.* $\varphi_1 \,\mathsf{U}\, \varphi_2$. Since this suffix premise is *wrt.* to the same composite formula $\varphi_1 \,\mathsf{U}\, \varphi_2$, it may well be the case that pUnt2 is applied again for suffix t. In fact, satisfaction proofs for until formulas are characterised by a series of pUnt2 applications, followed by an application of rule pUnt1 (the satisfaction proofs for $\varphi_1 \,\mathsf{R}\, \varphi_2$ and violation

proofs for $\varphi_1 \mathsf{U} \varphi_2$ and $\varphi_1 \mathsf{R} \varphi_2$ follow an analogous structure). This incremental analysis structure mirrors that of RV algorithms for LTL [Gei01,SRA04,BLS11] and contrasts with the descriptive nature of the *resp.* semantic definition for $\varphi_1 \mathsf{U} \varphi_2$ (Fig. 1) (which merely stipulates the *existence* of some index j at which point φ_2 holds without stating *how* to find this index).

We note the inherent symmetry between the satisfaction and violation rules, internalising the negation-propagation mechanism of the normalisation function $\langle - \rangle$ of §2 through rules pNeg and nNeg. For instance, there are no satisfaction (*resp.* violation) proof rules for the formula ff (*resp.* tt). The *resp.* predicate axioms for satisfactions and violations are dual to one another, as are the rules for conjunctions and disjunctions. More precisely, following $\langle \neg(\psi_1 \wedge \psi_2) \rangle \stackrel{\text{def}}{=} \langle \neg\psi_1 \rangle \vee \langle \neg\psi_2 \rangle$ from Fig. 1, the violation rules for conjunctions (nAnd1 and nAnd2) have the same structure as the satisfaction rules for the *resp.* disjunctions (pOr1 and pOr2). The symmetric structure carries over to the temporal proof rules as well, *e.g.*, violation rules nUnt1 and nUnt2 have an analogous structure to that of rules pRel1 and pRel2.

Example 2. Recall property $\mathsf{g}\ \mathsf{U}\ \mathsf{o}$ from Ex. 1. We can construct the satisfaction proof for trace go and the violation proof for trace gr below:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{g}(g)}\,\textsc{pPrd}}{go \vdash^+ \mathsf{g}}\,\textsc{pUnt2} \qquad \dfrac{\dfrac{}{\mathsf{o}(o)}\,\textsc{pPrd}}{o \vdash^+ \mathsf{o}}\,\textsc{pUnt1}}{go \vdash^+ \mathsf{g}\ \mathsf{U}\ \mathsf{o}} \qquad \dfrac{\dfrac{\dfrac{}{\overline{\mathsf{g}}(r)}\,\textsc{nPrd}}{r \vdash^- \mathsf{g}} \qquad \dfrac{\dfrac{}{\overline{\mathsf{o}}(r)}\,\textsc{nPrd}}{r \vdash^- \mathsf{o}}\,\textsc{nUnt1}}{r \vdash^+ \mathsf{g}\ \mathsf{U}\ \mathsf{o}} \qquad \dfrac{\dfrac{}{\overline{\mathsf{o}}(g)}\,\textsc{nPrd}}{gr \vdash^- \mathsf{o}}\,\textsc{nUnt2}}{gr \vdash^- \mathsf{g}\ \mathsf{U}\ \mathsf{o}}$$

Crucially, however, we are *unable* to construct *any* proof for the trace gg. For instance, attempting to construct a satisfaction proof fails because we hit the end-of-trace, ϵ, before completing the proof tree. Intuitively, we do not have enough information from the trace generated thus far to conclude that the complete trace satisfies the property. For instance, the next state may be o, in which case we can infer satisfaction for ggo, or it can be r, in which case we infer a violation for ggr; if it is g, we postpone any conclusive judgement once again.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{g}(g)}\,\textsc{pPrd}}{gg \vdash^+ \mathsf{g}}\,\textsc{pUnt2} \qquad \dfrac{\dfrac{\dfrac{}{\mathsf{g}(g)}\,\textsc{pPrd}}{g \vdash^+ \mathsf{g}} \qquad \dfrac{}{\epsilon \vdash^+ \mathsf{g}\ \mathsf{U}\ \mathsf{o}}\,??}{g \vdash^+ \mathsf{g}\ \mathsf{U}\ \mathsf{o}}\,\textsc{pUnt2}}{gg \vdash^+ \mathsf{g}\ \mathsf{U}\ \mathsf{o}}$$

■

Properties. Our proof system is sound, in the following sense.

Theorem 1 (Soundness). *For arbitrary t, φ:*

$$(t \vdash^+ \varphi \ \textit{implies}\ \forall s.\, ts \in [\![\varphi]\!]) \quad \textit{and} \quad (t \vdash^- \varphi \ \textit{implies}\ \forall s.\, ts \notin [\![\varphi]\!])$$

Proof. By rule induction on $t \vdash^+ \varphi$ and $t \vdash^- \varphi$. □

Example 3. The satisfaction and violation proofs of Ex. 2 suffice to prove $gos \in [\![\mathsf{g}\ \mathsf{U}\ \mathsf{o}]\!]$ and $grs \notin [\![\mathsf{g}\ \mathsf{U}\ \mathsf{o}]\!]$ for any (infinite) suffix s. Moreover, to determine whether $r\ldots \notin [\![(\neg\mathsf{r}) \wedge \mathsf{X}\mathsf{r}]\!]$ from Ex. 1, it suffices to consider the prefix r and

either construct a violation proof directly, or else normalise the negation of the formula, $\langle\neg((\neg\mathsf{r}) \wedge \mathsf{X}\mathsf{r})\rangle = \mathsf{r} \vee \mathsf{X}\bar{\mathsf{r}}$ and construct a satisfaction proof:

$$\textsc{nAnd1}\ \dfrac{\textsc{pNeg}\ \dfrac{\textsc{pPre}\ \dfrac{\mathsf{r}(r)}{r\ \vdash^{+}\ \mathsf{r}}}{r\ \vdash^{-}\ \neg\mathsf{r}}}{r\ \vdash^{-}\ (\neg\mathsf{r}) \wedge \mathsf{X}\mathsf{r}} \qquad \textsc{pOr1}\ \dfrac{\textsc{pPre}\ \dfrac{\mathsf{r}(r)}{r\ \vdash^{+}\ \mathsf{r}}}{r\ \vdash^{+}\ \mathsf{r} \vee \mathsf{X}\bar{\mathsf{r}}}$$ ■

Remark 1. The apparent redundancy (Ex. 3) allows us to use the proof system as a unifying framework that embeds other approaches (*cf.* §5), which may handle negation directly [SRA04], or work exclusively with formulas in nnf [Gei01].

Our proof system handles empty strings ϵ, as these arise naturally from the incremental analysis of finite traces discussed above.

Example 4. We can prove $oo\ldots \in [\![\mathsf{X}\,\mathsf{X}\,\mathsf{tt}]\!]$ from the prefix oo, by constructing the proof tree below; the leaf node relies on being able to deduce $\epsilon \vdash^{+} \mathsf{tt}$:

$$\textsc{pNxt}\ \dfrac{\textsc{pNxt}\ \dfrac{\textsc{pTru}\ \dfrac{}{\epsilon\ \vdash^{+}\ \mathsf{tt}}}{o\ \vdash^{+}\ \mathsf{X}\,\mathsf{tt}}}{oo\ \vdash^{+}\ \mathsf{X}\,\mathsf{X}\,\mathsf{tt}}$$ ■

Theorem 2 (Incompleteness). *For arbitrary t, φ:*

$$(\forall s.\, ts \in [\![\varphi]\!] \textit{ does not imply } t \vdash^{+} \varphi) \quad \textit{and} \quad (\forall s.\, ts \notin [\![\varphi]\!] \textit{ does not imply } t \vdash^{-} \varphi)$$

Proof. By counter example. For the positive case, consider $t = \epsilon$. We have $\forall s.\, ts \in [\![\mathsf{X}\,\mathsf{tt}]\!]$ but $t \not\vdash^{+} \mathsf{X}\,\mathsf{tt}$, $\forall s.\, ts \in [\![p \vee \bar{p}]\!]$ but $t \not\vdash^{+} p \vee \bar{p}$, and $\forall s.\, ts \in [\![\mathsf{ff}\ \mathsf{R}\ \mathsf{tt}]\!]$ but $t \not\vdash^{+} \mathsf{ff}\ \mathsf{R}\ \mathsf{tt}$. Curiously, whenever $p(\sigma)$ holds for *all* $\sigma \in \Sigma$, we also have $\forall s.\, ts \in [\![p]\!]$ but $t \not\vdash^{+} p$. Analogous examples can be drawn up for the negative case. □

We can however prove completeness for a syntactic subset of the logic, limiting ourselves to *discriminating* predicates[3], *i.e.*, predicates p where $\exists \sigma_1, \sigma_2 \in \Sigma$ such that $\sigma_1 \neq \sigma_2, p(\sigma_1)$ and $\neg p(\sigma_2)$. We define the following syntactic subset:

$$\begin{aligned} \phi \in \textsc{pLtl} &::= \mathsf{tt} \mid \mathsf{ff} \mid p \mid \bar{p} \mid \phi_1 \wedge \phi_2 \mid \phi_1\ \mathsf{U}\ \phi_2 \mid \neg\gamma \\ \gamma \in \textsc{nLtl} &::= \mathsf{tt} \mid \mathsf{ff} \mid p \mid \bar{p} \mid \gamma_1 \vee \gamma_2 \mid \gamma_1\ \mathsf{R}\ \gamma_2 \mid \neg\phi \end{aligned}$$

Theorem 3 (Partial Completeness). *For arbitrary t, ϕ, γ:*

$$(\forall s.\, ts \in [\![\phi]\!] \quad \textit{implies} \quad t \vdash^{+} \phi) \quad \textit{and} \quad (\forall s.\, ts \notin [\![\gamma]\!] \quad \textit{implies} \quad t \vdash^{-} \gamma)$$

Proof. By induction on the structure of ϕ and γ. □

4 An Automation

An automated proof search using the rules in Fig. 2 can be *syntax directed* by the formula (and the polarity) since, for most formulas, there is only *one* applicable rule. Moreover, the exception cases have *at most* two applicable rules.

[3] This does not decrease expressivity, since tt and ff can be used for the other cases. Note also that the co-predicate of a discrimination predicate is also discriminating.

$$\mathbf{exp}(d) \stackrel{\text{def}}{=} \begin{cases} \{(\!|\,|\!)\} & \text{if } (\!|\,|\!) \in d \\ \{\} & \text{if } d = \{\} \\ d & \text{if } c \in d \text{ implies } \mathbf{sat}(c) \\ \mathbf{exp}(\bigcup_{c \in d} \mathbf{expC}(c)) & \text{otherwise} \end{cases}$$

$$\mathbf{expC}(c) \stackrel{\text{def}}{=} \bigoplus_{o \in c} \mathbf{expO}(o)$$

$$\mathbf{expO}(o) \stackrel{\text{def}}{=} \begin{cases} \{c \mid r \in \mathbf{rls}(\varphi, q), c = \mathbf{prm}(r, t, \varphi)\} & \text{if } o = (t, \varphi)^q \\ \{(\!|\, \lceil \epsilon, \varphi \rceil^q \,|\!)\} & \text{if } o = \lceil \epsilon, \varphi \rceil^q \end{cases}$$

Fig. 3. A breadth-first incremental search algorithm

Notation. In what follows, $(t, \varphi)^+$ and *resp.* $(t, \varphi)^-$ denote the *resp.* outstanding proof obligations $t \vdash^+ \varphi$ and $t \vdash^- \varphi$. Since our algorithm works on partial traces, $\lceil \epsilon, \varphi \rceil^+$ and $\lceil \epsilon, \varphi \rceil^-$ are used to denote *saturated* proof obligations, where the string ϵ does not yield enough information to complete the proof search (*e.g.,* $\epsilon \vdash^+ \mathsf{g}\ \mathsf{U}\ \mathsf{o}$ in Ex. 2). A *conjunction set* $(\!|\, o_1, \ldots, o_n \,|\!)$ denotes a conjunction of proof obligations; metavariables o_i range over obligations of the form $(t, \varphi)^q$ or $\lceil t, \varphi \rceil^q$ for $q \in \{+, -\}$. A *disjunction set* $\{c_1, \ldots, c_n\}$, where c_i range over conjunction sets, denotes a disjunction of conjunction sets.[4] We employ a merge operation over disjunction sets, $\oplus$, defined below:

$$d \oplus d' \stackrel{\text{def}}{=} \{c \cup c' \mid c \in d, c' \in d'\}$$

The disjunction set $\{(\!|\,|\!)\}$ acts as the *identity*, , *i.e.,* $\{(\!|\,|\!)\} \oplus d = d \oplus \{(\!|\,|\!)\} = d$, whereas the disjunction set $\{\}$ *annihilates* such sets, *i.e.,* $\{\} \oplus d = d \oplus \{\} = \{\}$.

Algorithm. A breadth-first proof search algorithm is described in Fig. 3. Disjunction sets encode the alternative proof derivations that may lead to a completed proof-tree (resulting from multiple proof rules that can be applied at certain stages of the search), and conjunction sets represent the outstanding obligations within each potential derivation. Thus, a disjunction set with an element $(\!|\,|\!)$, denotes a *successful* search, whereas an empty disjunction set $\{\}$ represents a *failed* search. Another terminating condition for the search algorithm of Fig. 3 is when a disjunction set contains only *saturated* conjunction sets: these containing *only* saturated obligations of the form $\lceil \epsilon, \varphi \rceil^q$ (the predicate $\mathbf{sat}(c)$ denotes this).

To verify whether the judgement $t \vdash^q \varphi$ holds, we initiate the function $\mathbf{exp}(-)$ with the disjunction set $\{(\!|\, (t, \varphi)^q \,|\!)\}$. If none of the terminating conditions in Fig. 3 are met, $\mathbf{exp}(-)$ expands each conjunction set using $\mathbf{expC}(-)$, and recurses. Conjunction set expansion consists in expanding and merging *every* proof obligation using $\mathbf{expO}(-)$ and $\oplus$. Obligation expansion returns a disjunction set, where each conjunction set denotes the proof obligations resulting from the premises of the rules applied. It uses two auxilliary functions:

[4] For clarity, conjunction set notation, $(\!|\, - \,|\!)$, differs from that of disjunction sets, $\{-\}$.

– $\mathbf{rls}(\varphi, q)$ returns a set of rule names r from Fig. 2 that can be applied to obligations with the formula φ and polarity qualifier q (*e.g.*, $\mathbf{rls}(\varphi_1 \,\mathsf{U}\, \varphi_2, +) = \{\textsc{pUnt1}, \textsc{pUnt2}\}$ and $\mathbf{rls}(\mathsf{X}\varphi, -) = \{\textsc{nNxt}\}$).
– $\mathbf{prm}(r, t, \varphi)$ returns a conjunction set with the premises of rule r instantiated to the conclusion with string t and formula φ (*e.g.*, $\mathbf{prm}(\textsc{pUnt2}, go, \mathsf{g} \,\mathsf{U}\, \mathsf{o}) = (\!|\, (go, \mathsf{g})^+, (o, \mathsf{g} \,\mathsf{U}\, \mathsf{o})^+ \,|\!)$ and $\mathbf{prm}(\textsc{pTru}, go, \mathsf{tt}) = (\!|\,|\!)$). Importantly:
  (i) For cases such as $\mathbf{prm}(\textsc{pUnt2}, \epsilon, \mathsf{g} \,\mathsf{U}\, \mathsf{o})$ the function returns $(\!|\, \lceil \epsilon, \mathsf{g} \,\mathsf{U}\, \mathsf{o} \rceil^+ \,|\!)$ since the string ϵ prohibits the function from generating *all* the premises for the rule (one premise requires the string to be of length ≥ 1).
  (ii) The function is *undefined* when rule conditions are not satisfied (*e.g.*, $\mathbf{prm}(\textsc{pPrd}, g, \mathsf{o})$ is undefined since $\mathsf{o}(g)$ does not hold).

Example 5. Recall the inconclusive judgement $gg \vdash^+ \mathsf{g} \,\mathsf{U}\, \mathsf{o}$ from Ex. 2.

$$\begin{aligned}\mathbf{exp}(\{(\!|\, (gg, \mathsf{g}\,\mathsf{U}\,\mathsf{o})^+ \,|\!)\}) &= \mathbf{exp}(\{(\!|\, (gg, \mathsf{o})^+ \,|\!), (\!|\, (gg, \mathsf{g})^+, (g, \mathsf{g}\,\mathsf{U}\,\mathsf{o})^+ \,|\!)\}) \\ &= \mathbf{exp}(\{\} \cup (\{(\!|\,|\!)\} \oplus \{(\!|\, (g, \mathsf{o})^+ \,|\!), (\!|\, (g, \mathsf{g})^+, (\epsilon, \mathsf{g}\,\mathsf{U}\,\mathsf{o})^+ \,|\!)\}) \\ &= \mathbf{exp}(\{(\!|\, (g, \mathsf{o})^+ |\!), (\!|\, (g, \mathsf{g})^+, (\epsilon, \mathsf{g}\,\mathsf{U}\,\mathsf{o})^+ \,|\!)\}) = \{(\!|\, \lceil \epsilon, \mathsf{g}\,\mathsf{U}\,\mathsf{o} \rceil^+ \,|\!)\} \quad \blacksquare\end{aligned}$$

Properties. An execution of $\mathbf{exp}(\{(\!|\, (t, \varphi)^q \,|\!)\})$ may yield either of *three* verdicts. Apart from success, $\{(\!|\,|\!)\}$, meaning that a full proof tree was derived, the algorithm partitions negative results as either a definite fail, $\{\}$, or an *inconclusive* verdict, consisting of a saturate disjunction set d (where $c \in d$ implies $\mathbf{sat}(c)$).

Saturated disjunction sets make the algorithm *incremental*, in the following sense. When a further suffix t' is learnt to a judgement $t \vdash^q \varphi$ with an inconclusive verdict, we can *reuse* the saturated disjunction set returned for $t \vdash^q \varphi$, instead of processing $tt' \vdash^q \varphi$ from scratch. This is done by converting each obligation of the form $\lceil \epsilon, \varphi \rceil^q$ in each saturated conjunction set to the *active* obligation $(t', \varphi)^q$ using an auxilliary "append" function $\mathbf{app}(-)$.

Example 6. To determine whether $ggo \vdash^+ \mathsf{g} \,\mathsf{U}\, \mathsf{o}$ holds, we can take the inconclusive outcome of $\mathbf{exp}(\{(\!|\, (gg, \mathsf{g} \,\mathsf{U}\, \mathsf{o})^+ \,|\!)\})$ from Ex. 5, convert the saturated obligations using suffix o, $\mathbf{app}(\{(\!|\, \lceil \epsilon, \mathsf{g}\,\mathsf{U}\,\mathsf{o} \rceil^+ \,|\!)\}, o) = \{(\!|\, (o, \mathsf{g}\,\mathsf{U}\,\mathsf{o})^+ \,|\!)\}$, and calculate from that point onwards, $\mathbf{exp}(\{(\!|\, (o, \mathsf{g}\,\mathsf{U}\,\mathsf{o})^+ \,|\!)\}) = \{(\!|\,|\!)\}$. $\blacksquare$

Theorem 4 (Incrementality). $\boldsymbol{sat}(\boldsymbol{exp}(\{(\!|\, (t_1, \varphi)^q \,|\!)\}))$ *implies*

$$\boldsymbol{exp}(\{(\!|\, (t_1 t_2, \varphi)^q \,|\!)\}) = \boldsymbol{exp}(\boldsymbol{app}(\boldsymbol{exp}(\{(\!|\, (t_1, \varphi)^q \,|\!)\}), t_2))$$

Proof. By induction on t_2. $\square$

The algorithm of Fig. 3 is also *decidable* for the proof rules of Fig. 2. Intuitively, the main reason for this is because the proof system is *cut-free*, where rule premises are either defined in terms of string suffixes or subformula. Formally, we define a rank function $|-|$ mapping proof obligations to pairs of naturals, for which we assume a lexicographical ordering $(n_1, m_1) \geq (n_2, m_2) \stackrel{\text{def}}{=} n_1 \geq n_2 \vee (n_1 = n_2 \wedge m_1 \geq m_2)$ and the obvious function $\mathbf{max}(-)$ returning the

greatest element from a set of such pairs. Apart from $|t|$, we also assume $|\varphi|$ returning the *maximal depth* of the formula (*e.g.,* $|p \,\mathsf{U}\, \neg(\bar{p} \vee p)| = 3$ and $|\bar{p}| = 0$).

$$|(t,\varphi)^q| \stackrel{\text{def}}{=} (|t|,|\varphi|) \qquad |\lceil t,\varphi \rceil^q| \stackrel{\text{def}}{=} (0,0) \qquad |c| \stackrel{\text{def}}{=} \mathbf{max}(\{|o| \mid o \in c\} \cup \{(0,0)\})$$
$$|d| \stackrel{\text{def}}{=} \text{if } (\!|\,|\!) \in d \text{ then } (0,0) \text{ else } \mathbf{max}(\{|c| \mid c \in d\} \cup \{(0,0)\})$$

Above, the rank function maps saturated obligations to the bottom element $(0,0)$. We overload the function to conjunction sets, where we add $(0,0)$ to the $\mathbf{max}(-)$ calculation to cater for the case where c is empty. Following a similar pattern, we also extend the rank function to disjunction sets, but equate all sets with an empty conjunction set to the bottom element $(0,0)$; this mirrors the termination condition of the algorithm in Fig. 3 which terminates the search as soon as the shortest proof tree is detected.

Theorem 5 (Decidability). ***exp***$(\{(\!|\,(t,\varphi)^q\,|\!)\})$ *always terminates.*

Proof. Follows from the fact that when $|d| = (0,0)$, $\mathbf{exp}(d)$ terminates immediately, and when $|d| \neq (0,0)$, we have $|d| > |\bigcup_{c \in d} \mathbf{expC}(c)|$. □

Runtime Monitoring. We can obtain a setup akin to the three-valued monitors of [BLS11] with outcome **Y**, denoting satisfaction, outcome **N**, denoting violation, and outcome **?**, denoting an inconclusive outcome. Following [BLS11], given a finite trace t and a property φ, we attempt to construct a deduction for *both* the satisfaction, $t \vdash^+ \varphi$ and violation, $t \vdash^- \varphi$, by *concurrently* running $\mathbf{exp}(\{(\!|\,(t,\varphi)^+\,|\!)\})$ and $\mathbf{exp}(\{(\!|\,(t,\varphi)^-\,|\!)\})$ with the following possible outcomes:

1. We are able to construct a proof for $t \vdash^+ \psi$, corresponding to **Y**.
2. We are able to construct a proof for $t \vdash^- \psi$, corresponding to **N**.
3. We are unable to construct proofs for either case, corresponding to **?**.

Remark 2. The fourth possible outcome, *i.e.,* constructing a proof for *both* $t \vdash_3^+ \psi$ and $t \vdash_3^- \psi$, is ruled out by soundness (Thm. 1), which implicitly guarantees that our analysis is consistent (since the semantics is defined in terms of sets).

Like in most online RV setups, Thm. 4 allows for *incremental* monitor, as soon as individual trace elements are received. Moreover, Thm. 5 allows for a *safe synchronous* instrumentation, where the monitor and system execute in lock-step (*i.e.,* the system is paused after producing each monitored event so as to allow the monitor to carry out its analysis and perform timely detections). Since the monitoring analysis always terminates, the monitored system is guaranteed to be able to progress normally under a synchronous instrumentation.

5 Alternative RV Symbolic Techniques for LTL

We relate our deductive system to two prominent, but substantially distinct, symbolic techniques for LTL in the context of RV, namely [Gei01] and [SRA04].

5.1 Informative Prefixes

Intuitively, an *informative prefix* for a formula *explains* why a trace satisfies that formula [KYV01]. In [Gei01], trace satisfactions are monitored *wrt.* LTL *formulas in nnf*, by checking whether a trace contains an informative prefix.

Example 7. Recall $\mathsf{g}\ \mathsf{U}\ \mathsf{o}$ (Ex. 1). Prefix go is informative because (i) although the head, g, does not satisfy $\mathsf{g}\ \mathsf{U}\ \mathsf{o}$ in a definite manner, it allows the possibility of its suffix to satisfy the formula conclusively ($\mathsf{g}(g)$ holds); (ii) the immediate suffix, o, satisfies $\mathsf{g}\ \mathsf{U}\ \mathsf{o}$ conclusively ($\mathsf{o}(o)$ holds). In [Gei01], both go and o are deemed to be *locally-informative wrt.* $\mathsf{g}\ \mathsf{U}\ \mathsf{o}$ but go generates satisfaction obligations for the immediate suffix (*temporal informative successor*). ■

The algorithm in [Gei01] formalises the notion of locally informative by converting formulas to their *informative normal forms*. Moreover, temporal informative successors are formalised through the function $\mathit{next}(-)$, returning a set of formulas from a given formula and a trace element. For instance, in Ex. 7 $\mathit{next}(g, \mathsf{g}\ \mathsf{U}\ \mathsf{o}) = \{\mathsf{g}\ \mathsf{U}\ \mathsf{o}\}$ whereas $\mathit{next}(o, \mathsf{g}\ \mathsf{U}\ \mathsf{o}) = \{\}$. These functions are then used to construct automata that check for these properties over string prefixes.

$$\textsc{gTru}\frac{}{\mathit{linf}(t, \mathsf{tt}, \emptyset)} \qquad \textsc{gPre1}\frac{p(\sigma)}{\mathit{linf}(\sigma t, p, \emptyset)} \qquad \textsc{gPre2}\frac{\bar{p}(\sigma)}{\mathit{linf}(\sigma t, \bar{p}, \emptyset)}$$

$$\textsc{gOr1}\frac{\mathit{linf}(t, \varphi_1, m)}{\mathit{linf}(t, \varphi_1 \vee \varphi_2, m)} \qquad \textsc{gOr2}\frac{\mathit{linf}(t, \varphi_2, m)}{\mathit{linf}(t, \varphi_1 \vee \varphi_2, m)}$$

$$\textsc{gAnd}\frac{\mathit{linf}(t, \varphi_1, m_1) \quad \mathit{linf}(t, \varphi_2, m_2)}{\mathit{linf}(t, \varphi_1 \wedge \varphi_2, m_1 \cup m_2)} \qquad \textsc{gNxt}\frac{}{\mathit{linf}(t, \mathsf{X}\varphi, \{\varphi\})}$$

$$\textsc{gUnt1}\frac{\mathit{linf}(t, \varphi_2, m)}{\mathit{linf}(t, \varphi_1\ \mathsf{U}\ \varphi_2, m)} \qquad \textsc{gUnt2}\frac{\mathit{linf}(t, \varphi_1, m)}{\mathit{linf}(t, \varphi_1\ \mathsf{U}\ \varphi_2, m \cup \{\varphi_1\ \mathsf{U}\ \varphi_2\})}$$

$$\textsc{gRel1}\frac{\mathit{linf}(t, \varphi_1, m_1) \quad \mathit{linf}(t, \varphi_2, m_2)}{\mathit{linf}(t, \varphi_1\ \mathsf{R}\ \varphi_2, m_1 \cup m_2)} \qquad \textsc{gRel2}\frac{\mathit{linf}(t, \varphi_2, m)}{\mathit{linf}(t, \varphi_1\ \mathsf{R}\ \varphi_2, m \cup \{\varphi_1\ \mathsf{R}\ \varphi_2\})}$$

In this section, we express the locally-informative predicate and the associated temporal informative successors as the single judgement $\mathit{linf}(t, \varphi, m)$, defined as the least relation satisfying the rules above. It states that t is locally informative for φ with obligations $m \in \mathcal{P}(\textsc{eLtl})$ for the succeeding suffix. For example, for a formula $\mathsf{X}\varphi$, *any* string t is locally informative, but requires the immediate suffix to satisfy φ (see $\textsc{gNxt}$); in the case of $\varphi_1\ \mathsf{U}\ \varphi_2$, if t is locally informative for ψ_1 with suffix obligations m, then t is also locally informative for $\varphi_1\ \mathsf{U}\ \varphi_2$ with obligations $m \cup \{\varphi_1\ \mathsf{U}\ \varphi_2\}$ (see $\textsc{gUnt2}$). Informative prefixes are formalised as the predicate $\mathit{inf}(t, \varphi)$ below. Note that recursion in $\mathit{inf}(t, \varphi)$ is employed on a substring of t and terminates when $m = \emptyset$.

$$\mathit{inf}(t, \varphi) \stackrel{\text{def}}{=} \exists m.\ \Big(\mathit{linf}(t, \varphi, m) \quad \text{and} \quad \big(\varphi' \in m \text{ implies } (\mathit{inf}([t]^1, \varphi'))\big)\Big)$$

Example 8. We can deduce that $\mathit{inf}(go, \mathsf{g}\ \mathsf{U}\ \mathsf{o})$ because $\mathit{linf}(go, \mathsf{g}\ \mathsf{U}\ \mathsf{o}, \{\mathsf{g}\ \mathsf{U}\ \mathsf{o}\})$ and then that $\mathit{linf}(o, \mathsf{g}\ \mathsf{U}\ \mathsf{o}, \emptyset)$. ■

We can formally show a correspondence between informative prefixes and our monitoring proof systems.

Theorem 6. *For all φ in nnf, $\mathit{inf}(t, \varphi)$ iff $t \vdash^{+} \varphi$*

Proof. By structural induction on t, then by rule induction on $\mathit{linf}(t, \varphi, m)$ for the *only-if* case. By rule induction for the *if* case. □

In the *only-if* direction, Thm. 6 ensures that our system is as expressive as [Gei01]. In the *if* direction, Thm. 6 shows that every derivation in our proof system corresponds to an informative prefix as defined in [KYV01][5]. This reinforces existing justifications as to why our system is unable to symbolically process certain prefix and formula pairs.

Example 9. The proof system of §3 is unable to deduce $\epsilon \vdash^{+} \mathsf{Xtt}$ (*cf.* proof for Thm. 2) even though, on a semantic level, this holds for any string continuation because ϵ in *not* an informative prefix of Xtt. ■

Thm. 6 has another important implication. One justification of informative prefixes is that any bad/good prefixes detected can be accompanied by an explanation [BLS11]. However, whereas in [Gei01] this explanation is given in terms of the algorithm implementation, delineating the proof system from its implementing monitor (as in our case) allows for better separation of concerns, by giving the explanation as a derivation[6] in terms of the proof rules of Fig. 2.

5.2 Derivatives

In a derivatives approach [HR01,SRA04], LTL formulas are interpreted as *functions* that take a state *i.e.,* an element of the alphabet Σ, and return another LTL formula. The returned formula is then applied again to the next state in the trace, until either one of the *canonical* formulas tt or ff are reached; the trace analysis stops at canonical formulas, since tt (*resp.* ff) are idempotent, returning tt (*resp.* ff), irrespective of the state applied to. In [SRA04], coinductive deductive techniques are used on derivatives to establish LTL formula equivalences, which are then used to obtain optimal monitors for good/bad prefixes.

Example 10. Recall $\epsilon \nvdash^{+} \mathsf{Xtt}$ from Ex. 9. In [SRA04], they establish that formulas tt and Xtt are equivalent *wrt.* good prefixes, $\mathsf{tt} \equiv_G \mathsf{Xtt}$, which allows them to reason symbolically about ϵ and Xtt in terms of ϵ and tt instead. ■

Formally, a derivative is a rewriting operator $\_\{\_\} :: \textsc{aLtl} \times \Sigma \longrightarrow \textsc{aLtl}$ (adapted from [SRA04]) defined on the structure of the formula as follows:

[5] As a corollary, we also establish a correspondence between $t \vdash^{-} \varphi$ and $\mathit{inf}(t, \neg\varphi)$ as used in [Gei01] for bad prefixes.

[6] The algorithm in §4 can be easily extended so as to record the rules used.

$$\mathsf{tt}\{\sigma\} \stackrel{\text{def}}{=} \mathsf{tt} \qquad \mathsf{ff}\{\sigma\} \stackrel{\text{def}}{=} \mathsf{ff}$$
$$\mathsf{p}\{\sigma\} \stackrel{\text{def}}{=} \mathit{if}\ \mathsf{p}(\sigma)\ \mathit{then}\ \mathsf{tt}\ \mathit{else}\ \mathsf{ff}$$
$$(\neg\psi)\{\sigma\} \stackrel{\text{def}}{=} (\mathsf{tt} \oplus \psi)\{\sigma\} \qquad \psi_1 \oplus \psi_2\{\sigma\} \stackrel{\text{def}}{=} \psi_1\{\sigma\} \oplus \psi_2\{\sigma\}$$
$$\psi_1 \wedge \psi_2\{\sigma\} \stackrel{\text{def}}{=} \psi_1\{\sigma\} \wedge \psi_2\{\sigma\} \qquad \psi_1 \vee \psi_2\{\sigma\} \stackrel{\text{def}}{=} \psi_1\{\sigma\} \vee \psi_2\{\sigma\}$$
$$\mathsf{X}\psi\{\sigma\} \stackrel{\text{def}}{=} \psi$$
$$\psi_1\ \mathsf{U}\ \psi_2\{\sigma\} \stackrel{\text{def}}{=} \psi_2\{\sigma\} \vee (\psi_1\{\sigma\} \wedge \psi_1\mathsf{U}\psi_2)$$

Above, we position rewriting definitions for core LTL formulas of Fig. 1 on the left; formula rewriting however also uses an *extended* set of formulas that include falsity, ff, disjunction, $\psi_1 \vee \psi_2$, and exclusive-or, $\psi_1 \oplus \psi_2$. The derivatives algorithm also works up to formula normalisations using the following equalities:

$$\mathsf{tt} \wedge \psi \equiv \psi \qquad \mathsf{ff} \wedge \psi \equiv \mathsf{ff} \qquad \mathsf{ff} \vee \psi \equiv \psi \qquad \mathsf{tt} \vee \psi \equiv \mathsf{tt}$$
$$\psi \wedge \psi \equiv \psi \qquad \psi \vee \psi \equiv \psi \qquad \mathsf{ff} \oplus \psi \equiv \psi \qquad (\psi_1 \wedge \psi_2) \oplus \psi_1 \oplus \psi_2 \equiv \psi_1 \vee \psi_2$$

Thus, for any finite trace t of the form $\sigma_1\sigma_2\ldots\sigma_n$ we say:

- t is a good prefix for ψ iff $((\psi\{\sigma_1\})\{\sigma_2\})\{\ldots\sigma_n\} \equiv \mathsf{tt}$;
- t is a bad prefix for ψ iff $((\psi\{\sigma_1\})\{\sigma_2\})\{\ldots\sigma_n\} \equiv \mathsf{ff}$.

Example 11. The partial trace *go* is a *good prefix* for gUo (Ex. 1) because:

$$\begin{aligned}(\mathsf{gUo}\{g\})\{o\} &\stackrel{\text{def}}{=} \mathsf{o}\{g\} \vee \big(\mathsf{g}\{g\} \wedge \mathsf{gUo}\big)\{o\} \\ &\stackrel{\text{def}}{=} \quad \mathsf{ff} \ \vee \big(\mathsf{g}\{g\} \wedge \mathsf{gUo}\big)\{o\} \ \equiv\ \big(\mathsf{g}\{g\} \wedge \mathsf{gUo}\big)\{o\} \\ &\stackrel{\text{def}}{=} \big(\mathsf{tt} \wedge \mathsf{gUo}\big)\{o\} \ \equiv\ \mathsf{gUo}\{o\} \\ &\stackrel{\text{def}}{=} \mathsf{o}\{o\} \vee \big(\mathsf{g}\{o\} \wedge \mathsf{gUo}\big) \ \stackrel{\text{def}}{=}\ \ \mathsf{tt} \vee \big(\mathsf{g}\{o\} \wedge \mathsf{gUo}\big) \ \equiv\ \mathsf{tt}\end{aligned}$$ ■

Good (*resp.* bad) prefixes, as defined in[SRA04], correspond to finite traces with a satisfaction (*resp.* violation) proof in our system (§3), and vice-versa.

Theorem 7. *For any finite trace $t = \sigma_1\ldots\sigma_n$, and core LTL formula ψ:*

$$\big((\psi\{\sigma_1\})\{\ldots\sigma_n\} \equiv \mathsf{tt} \quad \textit{iff} \quad t \vdash^+ \psi\big) \quad \textit{and} \quad \big((\psi\{\sigma_1\})\{\ldots\sigma_n\} \equiv \mathsf{ff} \quad \textit{iff} \quad t \vdash^- \psi\big)$$

Proof. For the *only-if* case both statements are proved simultaneously by numerical induction on n and then by structural induction on ψ. For the *if* case both statements are proved simultaneously by rule induction. □

Apart from establishing a one-to-one correspondence between derivative prefixes and proof deductions for core LTL formulas in our system, Thm. 7 (together with Thm. 6) allows us to relate indirectly the informative prefixes of §5.1 to derivative prefixes. Moreover, Thm. 7 identifies from where the additional expressivity of the analysis in [SRA04] derives, namely through the deductive system for formula equivalence *wrt.* good/bad prefixed, $\vdash \psi_1 \equiv_G \psi_2$ and $\vdash \psi_1 \equiv_B \psi_2$. This opens up the possibility of merging the two approaches, perhaps by extending our deductive system with rules analogous to those show below:

$$\textsc{pEq}\frac{t \vdash^+ \varphi_1 \qquad \vdash \varphi_1 \equiv_G \varphi_2}{t \vdash^+ \varphi_2} \qquad \textsc{nEq}\frac{t \vdash^- \varphi_1 \qquad \vdash \varphi_1 \equiv_B \varphi_2}{t \vdash^- \varphi_2}$$

6 Conclusion

We presented a proof system and the respective monitor generation for runtime-verifying LTL properties. One novel aspect of our approach is that we tease apart the specification of the symbolic analysis from its automation. This allows us to localise correctness results *e.g.,* soundness is determined for the proof system, independent of the subsequent automation. The higher level of abstraction used elucidates completeness studies, seldom tackled in other work in RV *e.g.,* the syntactic subclass identified for Thm. 3 appears to be new. This separation of concerns also facilitates comparisons and cross-fertilisation with other symbolic techniques (§ 5) and leads to modular organisations that are easier to maintain *e.g.,* more efficient monitor automation may be considered without changing the proof rules. The concrete contributions are:

1. A *sound*, *local* LTL proof system inferring complete trace inclusion from finite prefixes, Thm. 1, together with completeness results, Thm. 2 and Thm. 3.
2. A mechanisation of proof derivations for this system that is formally incremental, Thm. 4, and decidable, Thm. 5.
3. An exposition of how the proof system can be used as a unifying framework where to relate different runtime monitoring formalisms, Thm. 7 and Thm. 6.

Related Work. Apart from the deductive system for LTL formula equivalence in [SRA04], there are other LTL proof systems [GPSS80,MP91,KI11,BL08]. Each differ substantially from ours. For instance, the model used in [GPSS80,MP91] is different from ours, *i.e.,* programs (*sets of traces*) instead of traces; the work in [GPSS80,KI11] is concerned with developing tableau methods for inferring the validity of a formula from a conjunction of formulas; [BL08] study cut-free sequent systems; importantly, none of these proof systems are local. In [MP91], they develop three *tailored* proof systems for separate classes of properties, namely safety, response and reactivity properties; crucially however, they do not consider aspects such as deductions from *partial* traces.

A substantial body of work studies alternative LTL semantics for partial traces [EFH$^+$03,BLS07,BLS11]; consult [BLS10] for a comprehensive survey. Although complementary, the aim and methodology of this work is substantially different from ours. In particular, we keep the LTL semantics *constant*, and explore the soundness, completeness and expressivity aspects of our symbolic analysis *wrt.* to this fixed semantics.

Future Work. It would be fruitful to relate other LTL symbolic analyses to the ones discussed in §5. Our work may also be used as a point of departure for developing proof systems for other interpretations of LTL. For instance, a different LTL model to that of §2, consisting of both *finite* and infinite traces, alters the negation propagation identities used for the translation function (*e.g.,* $\neg \mathsf{X}\, \psi \equiv \mathsf{X}\, \neg \psi$ does not hold) which, amongst other things, would require tweaking to the proof rules. Similar issues arise in distributed LTL interpretations such as [BF12] where instead of having one execution trace, we have a *set of traces*

(one for each location). We also leave complexity analysis and the assessment of the runtime overheads introduced by our setup as future work.

References

[BF12] Andreas Bauer and Ylis Falcone. Decentralised LTL Monitoring. In *FM*, volume 7436 of *LNCS*, pages 85–100. Springer, 2012.

[BL08] Kai Brunnler and Martin Lange. Cut-free sequent systems for temporal logic. *JLAP*, 76(2):216 – 225, 2008.

[BLS07] Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *RV*, volume 4839 of *LNCS*, pages 126–138, Berlin, Heidelberg, November 2007. Springer.

[BLS10] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *Logic and Comput.*, 20(3):651–674, 2010.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *TOSEM*, 20(4):14, 2011.

[BS92] Julian Bradfield and Colin Stirling. Local model-checking for infinite state spaces. *TCS*, 96:157–174, 1992.

[Bus98] Samuel R. Buss, editor. *Handbook of Proof Theory*. Elsevier, 1998.

[EFH+03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated pathsxs. In *CAV*, volume 2725 of *LNCS*, pages 27–39. Springer, 2003.

[Gei01] Marc Geilen. On the construction of monitors for temporal logic properties. *ENTCS*, 55(2):181–199, 2001.

[GPSS80] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *POPL*, pages 163–173, New York, NY, USA, 1980. ACM.

[HR01] Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *ASE*, pages 135–143, Wash., DC, USA, 2001. IEEE.

[KI11] Kensuke Kojima and Atsushi Igarashi. Constructive linear-time temporal logic: Proof systems and kripke semantics. *Inf. Comput.*, 209(12):1491–1503, 2011.

[KYV01] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Form. Methods Syst. Des.*, 19(3):291–314, October 2001.

[LS09] Martin Leucker and Christian Schallhart. A brief account of Runtime Verification. *JLAP*, 78(5):293 – 303, 2009.

[MP91] Zohar Manna and Amir Pnueli. Completing the Temporal Picture. *Theoretical Computer Science*, 83(1):97 – 130, 1991.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *SFCS*, pages 46–57, Wash., DC, USA, 1977. IEEE.

[RH05] Grigore Roşu and Klaus Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engg.*, 12(2):151–197, April 2005.

[SRA04] Koushik Sen, Grigore Rosu, and Gul Agha. Generating optimal linear temporal logic monitors by coinduction. In *ASIAN*, LNCS, pages 260–275. Springer, 2004.

[SW91] Colin Stirling and David Walker. Local model-checking in the modal mu-calculus. *TCS*, 89:161–177, 1991.

[TS00] A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 2000.